



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F	A2	(11) International Publication Number: WO 99/41651 (43) International Publication Date: 19 August 1999 (19.08.99)
(21) International Application Number: PCT/SG98/00010 (22) International Filing Date: 13 February 1998 (13.02.98) (71) Applicant (for all designated States except US): NATIONAL COMPUTER BOARD, acting through ITS R & D DIVISION, THE INFORMATION TECHNOLOGY INSTITUTE [SG/SG]; 11 Science Park, Science Park II, Singapore 117685 (SG). (72) Inventor; and (75) Inventor/Applicant (for US only): CHIANG, Kuo, Chiang [SG/SG]; Information Technology Institute, 11 Science Park, Singapore Science Park, Singapore 117685 (SG). (74) Agent: K. T. LIM & COMPANY; Tong Eng Building, 101 Cecil Street #25-06, Singapore 069533 (SG).		(81) Designated States: AU, CA, CN, GB, IL, JP, KR, NZ, SG, US. Published <i>Without international search report and to be republished upon receipt of that report.</i>
(54) Title: METHOD FOR PROTECTING BYTECODE (57) Abstract A method, software tool and system for protecting bytecode, which involves encrypting bytecode for an application, providing the encrypted bytecode to a user, executing a code loader to load the encrypted bytecode, access a decryption key, decrypt the encrypted bytecode, and pass the decrypted bytecode to a run-time system, and executing the decrypted bytecode with the run-time system.		

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

METHOD FOR PROTECTING BYTECODE

Field of the Invention

- 5 The present invention relates to a method for protecting compiled software application code, and a code protection system.

Background of Invention

- 10 There are many programming languages available that compile to platform independent bytecode, such as Lisp and Java, the latter being described in references A and B recited below. One of the greatest criticisms of such programming languages is that the bytecode, being defined at a high level, can easily be reverse-engineered into the source language. The problem is compounded with the wide
15 availability of bytecode decompilers on the Internet, as described in reference C. It means that any novice computer user can produce readable source code from compiled bytecode. While the output of bytecode decompilers may not be the same as the original source code, it is nevertheless sufficiently close to allow any programmer to understand and make unauthorised use of the output.

20

- The current most commonly used protection against bytecode decompilers is a bytecode obfuscator, as described in reference D. A bytecode obfuscator is a tool that goes through an application and renames all the symbolic information inside, such as the class names, method names and field names. Bytecode obfuscation does not
25 protect the bytecode from being decompiled, but it makes the decompiled code a lot more difficult to understand.

- Some bytecode obfuscators protect further by adding corrupted information into the application files to confuse the bytecode decompilers. This technique is frequently
30 applied at the expense of violating the standard bytecode file format specification, which may render the resulting bytecode incompatible with certain implementations

- 2 -

of the run-time system. This technique is also limited, due to the availability of free tools on the Internet that can remove the corrupted information, as described in reference E.

- 5 Certain other techniques for protecting bytecode have been discussed in reference F. These other techniques, however, fall short in that they either require the main application logic to reside on a separate machine, or they require the developer not to use bytecode for sensitive application logic.

10 Summary of the Invention

In accordance with the present invention there is provided a method of protecting bytecode, including:

- encrypting bytecode for an application;
- 15 providing the encrypted bytecode to a user;
- executing a code loader to load the encrypted bytecode, access a decryption key, decrypt the encrypted bytecode, and pass the decrypted bytecode to a run-time system; and
- executing said decrypted bytecode with said run-time system.

20

The present invention also provides a software tool stored on a computer readable storage medium, including:

- means for accessing bytecode;
- means for determining if the bytecode is encrypted;
- 25 means for accessing a decryption key;
- means for decrypting encrypted bytecode using the decryption key; and
- means for passing bytecode to a run-time system.

- The present invention further provides a software protection system, including:
- 30 means for accessing bytecode;
 - means for determining if the bytecode is encrypted;

- 3 -

means for accessing a decryption key;
means for decrypting encrypted bytecode using the decryption key; and
means for passing bytecode to a run-time system.

5 The present invention also provides a method for execution by a computer system, including:

accessing bytecode;
determining if the bytecode is encrypted;
accessing a decryption key;
10 decrypting encrypted bytecode using the decryption key; and
passing bytecode to a run-time system.

Brief Description of the Drawing

15 A preferred embodiment of the present invention is hereinafter described, by way of example only, with reference to the accompanying drawing, wherein:

Figure 1 is a flow diagram of a process executed by a code loader of the preferred embodiment.

20 Detailed Description of the Preferred Embodiment

A software protection system of the preferred embodiment uses a software tool, hereinafter referred to as deCaf™, which includes an encryption module, a code loader and a driver program. The system comprises at least one computer having
25 electronic memory used for storing, reading and executing the software tool, and a run-time system for executing bytecode of a software application. The system normally comprises a server computer and a user computer, where the tool is stored on the server with application bytecode, and the bytecode when encrypted is transmitted to the user's computer with the code loader and the driver program for storage and
30 execution on the user's computer. The code loader and the drive program can be transmitted separately and also can be prestored on a user's computer. The

- 4 -

encryption module is used to encrypt bytecode, which can then only be decrypted by the code loader. The driver program installs the code loader for a run-time system of a user's machine, instructs the code loader to decrypt the encrypted bytecode, and then causes the decrypted code to be executed on the user's machine. By applying
5 encryption to bytecode, deCaf™ is able to prevent bytecode decompilers from reading the bytecode. The decompilers are unable to decompile the encrypted bytecode without first decrypting it. To understand the encrypted bytecode format, normally this will require modification of a run-time system, but deCaf™ does not require any such modification, as it is able to rely on the code loader which can be applied in any run-
10 time system that supports the installation of a code loader by the software application to be executed. In particular, deCaf™ is able to encrypt Java software applications and then execute them on an unmodified Java run-time system, being the Java Virtual Machine.

15 The encryption module when executed, normally by a transmitting server, acts on the compiled bytecode of a software application and applies an encryption algorithm to produce encrypted bytecode. Any secured encryption algorithm can be used, although one which enhances the execution speed of the encryption module is preferred.

20 In any encryption algorithm, there is an encryption key as well as a decryption key. In an asymmetric encryption algorithm, the encryption and decryption keys are different. While in a symmetric encryption algorithm, both keys are the same. Regardless of which is used, the decryption key is distributed to the application user so the encrypted bytecode can be decrypted later and executed.

25

This key distribution is done in one of the following ways:

1. Using a trusted public-private key authentication infrastructure. In the presence of a trusted public-private key authentication infrastructure, the user's public
30 key is used to encrypt the application bytecode. Then, the private key can decrypt the bytecode on receipt. As such, no one else other than a legitimate

- 5 -

user can run the software. No actual key distribution is required since the user uses his own private key for decryption of the software.

2. Embedding the decryption key in the encrypted bytecode. In the event that an
5 infrastructure to support key distribution is not available, the decryption key can be hidden within the encrypted bytecode. The code loader will know this hidden location, so that the key can be extracted at run-time to decrypt the bytecode. To avoid disclosing the secret location of the key within the encrypted
10 bytecode, the location can be made random by having it dependent on a relatively random data pattern, such as the message digest of the application code. The customised code loader will normally be provided together with the encrypted bytecode. As such, this method of distributing the decryption key is not completely secure, since a hacker could go through the code of the code
15 loader to try and determine the location of the decryption key. Nevertheless, the task of doing so is labourious enough to deter most hackers. Thus this key distribution method is still useful when protecting less sensitive application bytecode.
3. Distributing the decryption key separately. Instead of embedding the decryption
20 key within the encrypted application, the decryption key is passed separately to the user. The encrypted application bytecode is distributed to the user who is then required to register his particulars before the decryption key is passed. This key distribution method has the same weakness as the former, in that a
25 hacker could go through the labourious task of attempting to understand the code loader, and determine how it applies the decryption key. As such, its usefulness is also confined to protecting less sensitive application bytecode. However, this method helps keep track of legitimate users, much like the serial
30 number required to install most commercial software. If an illegal copy of encrypted software is received, it can be traced back to its original legitimate user.

- 6 -

In some run-time environments, such as Java's, application software running in the environment is given the ability to install its own code loader. A code loader (known as the ClassLoader in Java) is a small piece of software that is responsible for loading the application code from a disk, a network or any other storage medium. DeCaf™
5 exploits this feature by providing it's own code loader to perform code decryption at run-time, hence omitting the need to modify the run-time system.

The code loader's role is to find and load the correct application bytecode, and then pass it to the run-time system. To provide a code loader that can understand the
10 encrypted bytecode, appropriate decryption routines are inserted and executed between the loading of the application code and handing it over to the run-time system. This ensures that the run-time system always receives bytecode it can execute.

15 The code loader of deCaf™, as shown in Figure 1, first receives a request to load a bytecode file, at step 2. When the run-time system needs to load a bytecode file, it passes the request to the code loader for execution. The code loader then looks for the bytecode file, at step 4. The bytecode file can reside anywhere in the local disk, or even in a network. In most run-time systems, the list of places to look for the
20 bytecode is given by a predefined variable which the user can change. In Java, this is given by the CLASS PATH environment variable. Once the application bytecode file is loaded, the custom code loader determines whether it is in encrypted form, at step 6. In deCaf™ a predefined constant of 0xDECAF0 is assigned to all encrypted bytecode. The loader determines the existence of this constant in order to decide
25 whether the bytecode is encrypted or not. If the loader determines that the bytecode is encrypted, operation proceeds to step 8, whereas otherwise operation proceeds to step 12. At step 8, the code loader loads the decryption key. Loading the decryption key will depend on how the decryption key is distributed. In a current implementation of deCaf™, the decryption key is encoded in the encrypted bytecode. The loader
30 accesses this information, and loads the decryption key accordingly. Once the decryption key is loaded, the custom code loader executes the required decryption

- 7 -

algorithm on the encrypted code, at step 10. As mentioned earlier, any secure encryption algorithm can be chosen. One particular encryption algorithm can be chosen, or the type of encryption algorithm used can be encoded in the encrypted bytecode. In the latter case, the code loader looks up the type of encryption algorithm
5 used, and applies the corresponding decryption algorithm. Once the bytecode has been decrypted, or if it is not encrypted in the first place, it is passed directly to the run-time system, at step 12 by the code loader.

The driver program of deCaf™ is used to tie the components of the tool together and
10 run the encrypted application. The driver program executes the following steps:

1. Install the code loader in the run-time system;
2. Instruct the code loader to load the encrypted application bytecode; and
3. Run the decrypted application.

15 All the above tasks are dependent on the run-time system to be used. To install the code loader in Java, a code loader object of deCaf™ is initiated. This is followed by extracting out the encrypted application bytecode, and passing it to the loadClass procedure of the code loader. Finally, the Java reflection API is used to invoke the main procedure of the decrypted application.

20

Many modifications will be apparent to those skilled in the art without departing from the scope of the present invention as herein described with reference to the accompanying drawing.

25

References

- A. James Gosling, Bill Joy, Guy Steele, The Java Language Specification, ISBN: 0-201-63451-1, Addison Wesley.
- 5
- B. Tim Lindholm, Frank Yellin, The Java Virtual Machine Specification, ISBN: 0-201-63452-X, Addison Wesley.
- C. Decompilers available on the Internet:
- 10
- Mocha, <http://www.brouhahn.com/~eric/computers/mocha.html>
 - WingDis, <http://www.wingsoft.com/windis.shtml>
 - Jasmine, <http://www.members.tripod.com/~SourceTec/>
 - DeJaVu, <http://www.isg.de/OEW/Java/>
 - Jad, <http://web.unicom.com.cy/~kpd/iad.html>
- 15
- D. Qusay H. Mahmoud, Protect your bytecodes from reverse engineering/decompilation. In *Java World*, <http://www.javaworld.com/javatips/iw~javatip221.html>.
- E. Unobfuscator example: Zelix KlassMaster, <http://www.zelix.com/klassmaster/index.html>.
- 20
- F. Scott Oaks. Protecting your bytecode. In *Java Report*, pgs. 86-88, November 1997.

- 9 -

CLAIMS

1. A method of protecting bytecode, including:
encrypting bytecode for an application;
5 providing the encrypted bytecode to a user;
executing a code loader to load the encrypted bytecode, access a decryption key, decrypt the encrypted bytecode, and pass the decrypted bytecode to a run-time system; and
executing said decrypted bytecode with said run-time system.
10
2. A method as claimed in claim 1, wherein the decryption key is encoded in the encrypted bytecode.
3. A method as claimed in claim 1, wherein the decryption key is a private key of
15 a user.
4. A method as claimed in claim 1, including registering particulars of the user to obtain said decryption key.
- 20 5. A method as claimed in claim 1, including executing a driver program to install the code loader for the run-time system, and cause the code loader to access the encrypted bytecode.
6. A method as claimed in claim 5, including providing the code loader and driver
25 program with the encrypted bytecode.
7. A method as claimed in claim 1, including providing the code loader with the encrypted bytecode.
- 30 8. A method as claimed in claim 6 or 7, wherein said providing step comprises transmitting from a network computer to a computer of the user.

- 10 -

9. A software tool stored on a computer readable storage medium, including:
means for accessing bytecode;
means for determining if the bytecode is encrypted;
means for accessing a decryption key;
5 means for decrypting encrypted bytecode using the decryption key; and
means for passing bytecode to a run-time system.
10. A software tool as claimed in claim 9, wherein the determining means relies on predetermined data stored in encrypted bytecode.
- 10 11. A software tool as claimed in claim 9, wherein the key accessing means accesses data to determine the storage location of the decryption key.
12. A software tool as claimed in claim 11, wherein the key accessing means
15 accesses data to determine the decryption algorithm to be used by the decrypting means.
13. A software tool as claimed in any one of claims 9 to 12, wherein a code loader comprises all of said means, and the tool further includes:
20 means for installing the code loader in the run-time system;
means for causing the code loader to access the bytecode; and
means for causing execution of the bytecode when passed to the run-time system.
- 25 14. A software tool as claimed in claim 12, including means for encrypting bytecode.
15. A software protection system, including:
means for accessing bytecode;
30 means for determining if the bytecode is encrypted;
means for accessing a decryption key;

- 11 -

means for decrypting encrypted bytecode using the decryption key; and
means for passing bytecode to a run-time system.

16. A software protection system as claimed in claim 15, wherein the determining
5 means relies on predetermined data stored in encrypted bytecode.

17. A software protection system as claimed in claim 15, wherein the key accessing
means accesses data to determine the storage location of the decryption key.

10 18. A software protection system as claimed in claim 17, wherein the key accessing
means accesses data to determine the decryption algorithm to be used by the
decrypting means.

19. A software protection system as claimed in any one of claims 15 to 18, wherein
15 a code loader comprises all of said means, and the system further includes:
means for installing the code loader in the run-time system;
means for causing the code loader to access the bytecode; and
means for causing execution of the bytecode when passed to the run-time
system.

20

20. A software protection system as claimed in claim 19, including means for
encrypting bytecode.

21. A method for execution by a computer system, including:
25 accessing bytecode;
determining if the bytecode is encrypted;
accessing a decryption key;
decrypting encrypted bytecode using the decryption key; and
passing bytecode to a run-time system.

30

22. A method as claimed in claim 21, wherein the determining step is executed by

- 12 -

searching for predetermined data stored in encrypted bytecode.

23. A method as claimed in claim 21, wherein the key accessing step includes accessing data to determine the storage location of the decryption key.

5

24. A method as claimed in claim 23, wherein the key accessing step includes accessing data to determine the decryption algorithm to be used by the decrypting means.

10 25. A method as claimed in any one of claims 21 to 24, wherein a code loader executes the accessing, determining, decrypting and passing steps, and the method further includes:

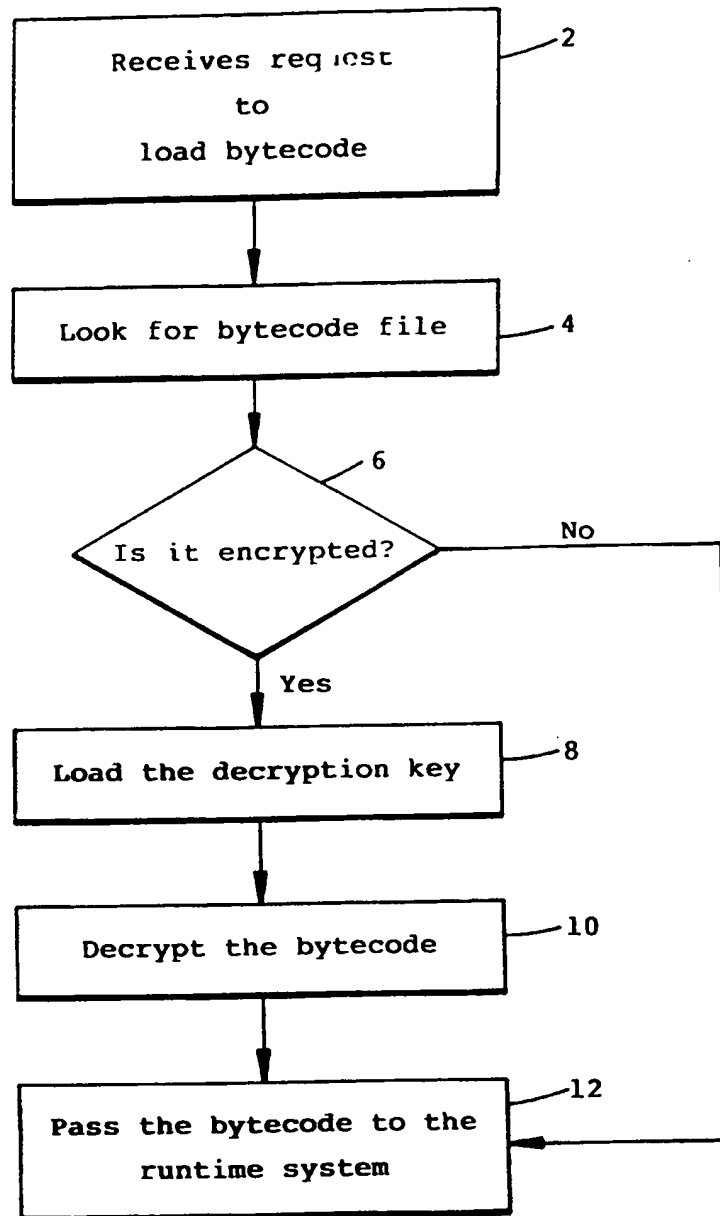
installing the code loader for the run-time system;

causing the code loader to access the bytecode; and

15 causing execution of the bytecode when passed to the run-time system.

26. A method as claimed in claim 25, including encrypting bytecode.

1/1

FIGURE 1

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER: _____**

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.